# Ejercicio de Reparaciones

**Objetivo**: El objetivo de esta práctica es la familiarización del alumno con la programación orientada a objetos en Java, con la generación de pruebas y con la herencia.

**Evaluación**: La práctica se podrá realizar en grupos de dos alumnos o de forma individual. La nota que se obtenga en esta práctica tendrá un peso que se especificará como parte de la nota de prácticas en la nota final de la asignatura. Se mantendrá el mismo enunciado de la práctica para la convocatoria extraordinaria.

**Entrega:** La práctica se entregará a través de la página web:

http://maui.ls.fi.upm.es/entrega/

La práctica entregada debe compilar en la versión 1.6 del J2SE de Oracle/Sun. En el momento de realizar la entrega, la práctica será sometida a una serie de pruebas que deberá superar para que la entrega sea admitida. El alumno dispondrá de **un número máximo de diez entregas**. Ahora bien, si el alumno realiza **más de cinco** entregas, se le **restará un punto** en la nota final de la práctica. Asimismo, por el hecho de que la práctica sea admitida, eso no implicará que la práctica esté aprobada.

**Grupos**: Los grupos estarán formados por dos alumnos. Los alumnos deben estar matriculados y dados de alta en el sistema de entrega (maui) antes de proceder a realizar el registro del grupo. Ninguno de los dos alumnos deberá haber realizado entrega alguna de la práctica antes de definir el grupo. El grupo se crea por medio de la URL:

http://maui.ls.fi.upm.es/entrega/CrearGruposProgramacion2.html

Una vez creado el grupo sólo podrá realizar la entrega el alumno que se pone en primer lugar en el grupo.

**Fecha límite**: Es el Jueves día **29 de Marzo** a las **10:00 AM horas**.

**Autoevaluación**: El alumno debe comprobar que su ejercicio no contiene ninguno de los errores explicados en el último apartado de este enunciado. **Si el ejercicio contiene alguno de estos errores, se calificará como suspenso**.

**Detección Automática de Copias**: Cada práctica entregada se comparará con el resto de prácticas entregadas en todos los grupos de la asignatura. Esto se realizará utilizando un sofisticado programa de detección de copias.

**Consecuencias de haber copiado**: Todos los alumnos involucrados en una copia, bien por copiar o por ser copiados, quedan inhabilitados para presentarse a todas las convocatorias de examen del presente curso, además de la posible apertura de expediente académico.

## Enunciado del problema

Se intentan modelizar distintos tipos de reparaciones (véase Figura 1). Una reparación tarda en realizarse un determinado número de ciclos dependiendo de su dificultad.

Se distinguen dos tipos de reparaciones, **normales** y **con prioridad**. Una reparación normal va a ser un tipo de reparación cuya realización puede tardar mucho tiempo en

realizarse. Por otro lado, una reparación con prioridad es un tipo de reparación que, al contrario que una reparación normal, tiene que realizarse pronto.

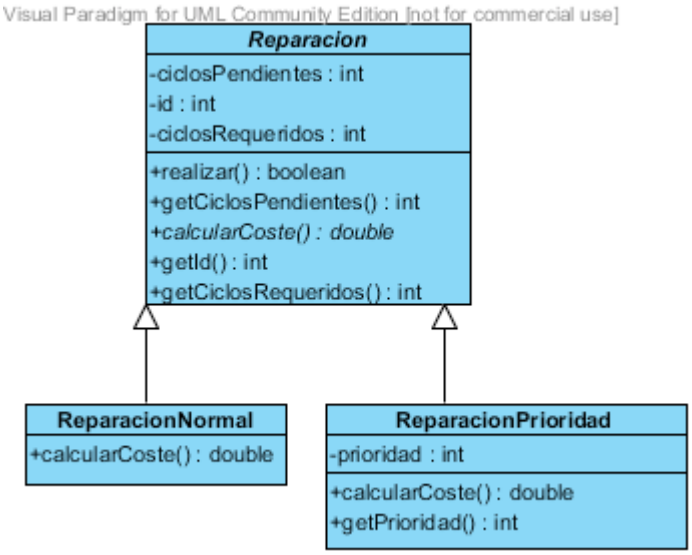


**Figura 1 Diagrama UML**

Una reparación con prioridad tendrá asociada una prioridad que representaremos mediante un valor numérico, en donde el **valor 1 representará la prioridad mínima**. Las reparaciones tienen un coste en función del número de ciclos que lleva la reparación, y la prioridad de la misma.

En la siguiente tabla se detallan los costes de las reparaciones según su tipo:

| **Tipo de Reparación** | **Coste en €** |
|---|---|
| R. con Prioridad P | 30*ciclos*P/2+25*ciclos/2+P*10 |
| R. Normal | 15*ciclos/3+25*2*ciclos/3 |

## Análisis orientado a objetos

La implementación se realizará con un proyecto "reparacion" (en un paquete que también se llamará "reparacion") que incluya las clases: "Reparacion", "ReparacionNormal" y "ReparacionPrioridad". Se facilita también un paquete "excepciones" y otro "pruebas" que contienen respectivamente la implementación de la excepciones que se van a usar en el proyecto y dos ficheros de prueba ("JUnitReparacionNormal" y "JunitReparacionPrioridad").

Dentro de cada objeto "Reparacion" se mantendrá su estado de progreso, que será decrementado en cada ciclo de trabajo. Cuando dicho progreso alcance el valor cero, eso significará que la reparación ha sido completada.

La clase "Reparacion" es una clase abstracta, de la que existen dos especializaciones: "ReparacionNormal" y "ReparacionPrioridad".

## Diseño de las clases

### Atributos de instancia:

**id**: número entero que representa un código identificador de una reparación concreta.

**ciclosRequeridos**: número entero que representa el número de ciclos de tiempo necesarios para realizar una reparación.

**ciclosPendientes**: número entero que representa el número de ciclos de tiempo que quedan pendientes para completar una reparación.

### Métodos:

**Reparacion**: constructor de la clase "Reparacion" que recibe como argumento el identificador de código de la reparación y el número de ciclos requeridos para su realización. Si el número de ciclos de una reparación en su creación es cero o negativo deberá producirse una excepción ErrorNumeroCiclosIncorrecto. En el caso de la subclase ReparacionPrioridad, el constructor debe recibir como argumentos, además de los ya mencionados, la prioridad y debe comprobar que es mayor o igual que 1. Si dicha prioridad fuera menor estrictamente que 1, debe lanzar la excepción ErrorPrioridadIncorrecta.

**getId**: método para recuperar el código de una reparación.

**getCiclosPendientes**: método para recuperar el número de ciclos que quedan pendientes en una reparación.

**getCiclosRequeridos**: método para recuperar el número total de ciclos requeridos para realizar completamente una reparación.

**realizar**: método que avanza un ciclo la reparación si quedan ciclos pendientes. Si no quedan ciclos pendientes, se lanza la excepción ErrorReparacionYaConcluida. Si quedan ciclos pendientes, además de avanzar un ciclo, se devuelve cierto si le siguen quedando ciclos a la reparación por realizarse, y falso si ya no quedan más ciclos para terminar la reparación (éste era el último ciclo que quedaba por realizarse).

**calcularCoste**: Método abstracto que determina el coste de la reparación. Este método se encuentra implementado en ReparacionNormal y ReparacionPrioridad de forma específica de acuerdo con la fórmula facilitada.

**getPrioridad**: método para recuperar la prioridad de una reparación. Solo se incluirá en la clase ReparacionPrioridad.

## Consideraciones de entrega

En este ejercicio se pide implementar la clase abstracta "Reparacion" y las dos clases derivadas "ReparacionNormal" y "ReparacionPrioridad".

Se entregarán tres ficheros **Reparacion.java**, **ReparacionNormal.java** y **ReparacionPrioridad.java** basados en los que se facilitan y sin comprimir. Se deben comentar las clases y los métodos.

### Errores graves a evitar:

1. Se declaran atributos públicos.
2. Utiliza atributos de clase o static (para esta práctica no son necesarios).
3. Se realizan operaciones de entrada/salida en alguna de las clases implementadas por el alumno, excepto en las ramas catch en donde sí que se permite.

### Pautas de programación a tener en cuenta:

1. La utilización de nombres significativos para los identificadores, así como la utilización de los convenios de Java.
2. La utilización de métodos auxiliares que implementen tareas comunes a varios métodos, y que de esa forma eviten la duplicidad de código.
3. La correcta indentación del código. Se recomienda la utilización en eclipse del atajo de teclado CTRL + i.
4. La implementación de un código eficiente que no realice operaciones innecesarias.